

# An Ox Class for Implementing Bootstrap Tests

James Davidson, University of Exeter

Version 3, August 2005

The Ox 3.x class `Bootstrap` provides the basic functions needed to set up parametric bootstrap tests and compute bootstrap confidence intervals. See, e.g. Horowitz (1999) for background. Procedures implemented in this version of the package include the regular bootstrap, the double bootstrap of Beran (1988), and the fast double bootstrap (Davidson and MacKinnon 2000). The double bootstrap procedure uses stopping rules proposed by Nankervis (2001) that much reduce its computational cost, particularly when the  $p$ -values are small. See Davidson (2003) for further details on these procedures, and an application. For cases where the resampled series contains heteroscedasticity of unknown form, the 'wild bootstrap' (Liu 1988, Mammen 1993) is implemented, using a choice of two-point distributions. The complete bootstrap distribution (i.e., the sorted replications) can be retrieved for further analysis.

The operations of randomly re-sampling residuals, generating the empirical distribution of the test statistic, computing  $p$ -values, and writing out the results, are performed by class member functions. `Simulation`, a class derived from `Bootstrap`, is also provided to facilitate Monte Carlo analysis of bootstrap tests.

The raw bootstrap  $p$ -values can be transformed by any increasing function on the unit interval, if this is tabulated in a matrix file. An alternative implementation of the double bootstrap is provided by letting this function be the  $p$ -value's own empirical distribution function (EDF) under  $H_0$ , tabulated by Monte Carlo simulation. More generally, this facility can be used for computing size-corrected powers.

Version 3 includes a block bootstrap option. Programs written for Versions 1 and 2 will run unchanged.

This package is offered freely to academic users on the following conditions:

1. Its use in published research is acknowledged. Please cite this document in your list of references.
2. Any modifications to the code, either bug-fixes or extensions of capability, are notified to the author and made available for distribution. Please send a copy to [james.davidson@exeter.ac.uk](mailto:james.davidson@exeter.ac.uk) with changes/additions clearly annotated.
3. (Disclaimer) The code is distributed "as is", with no warranties as to fitness for any purpose. Use it at your own risk.

Check for updates at <http://people.ex.ac.uk/jehd201/>

To implement the procedures, copy the files into your Ox directory and place the line

```
#include "btstrap.ox"
```

in your Ox 3.x program. Then create a derived class from one of the base classes, `Bootstrap` or `Simulation`, providing virtual functions to perform the model-specific operations. The main functions are required to perform the following tasks.

- Evaluate a statistic, or more generally an  $m \times 1$  vector of statistics, from a data set.
- Create a data set suitable for random resampling. In cases with i.i.d. data, this could be the original data set itself. More often, it involves estimating a model and returning the residuals.
- Create a bootstrap data set with generated or resampled data. This typically involves inverting the model previously estimated, to generate data from the resampled residuals.

Static variables should be declared to allow communication between these functions. See the examples in the accompanying .ox files, and the Ox documentation, for guidance on program structure.

## Bootstrap::Settings

```
Bootstrap::Settings(const iMeth, const aNames, const cReps,  
const OutFile, const iRanSeed, const cDReps, const dAlphaMax,  
const HFileIn);
```

**iMeth**            in: int, = 0 for the regular bootstrap test only (default);  
                  = 1 for the regular and double bootstrap test ( $m = 1$  only);  
                  = 2 for the regular and fast double bootstrap tests (FD1 and FD2);  
                  = 3 for all the tests ( $m = 1$  only);  
                  = 4 for quantiles of the bootstrap distribution;  
                  = -1 for test by tabulation only.

**aNames**            in:  $m \times 1$  array of strings, names for the test statistics.

**cReps**            in: int, the number of bootstrap replications.

**OutFile**          in: string, name and path of a file for text output.

**iRanSeed**        in: int, seed for the random number generator.

**cDReps**           in: int, the number of double bootstrap replications.

**dAlphaMax**       in: double, maximum double bootstrap  $p$ -value to be computed.

**HFileIn**          in: string, name and path of a file containing EDFs for size correction. The extension indicates the type of file, see the documentation of the Ox `loadmat` function.

*No return value.*

### Description

Used to select options for the run. All the arguments are optional, and if the default settings are desired, no call to `Settings` is needed. However, such arguments as appear must be strictly in the order shown, with 0s to set the default values.

- The strings in `aNames` should match the elements of the vector returned from `Estimate`. If omitted or set to 0, the default label "Statistic" is used.
- If `cReps` is omitted or set to 0, the default of 399 is used.
- If `OutFile` is omitted or set to 0, output is sent to the console only.
- If `iRanSeed` is omitted or set to 0, the random number generator is initialised with the current time from the system clock.
- `cDReps` is ignored unless `iMeth = 1` or 3. If omitted or set to 0, the default of `cReps` is used.
- `dAlphaMax` is ignored unless `iMeth = 1` or 3. If `dAlphaMax = 0`, it is reset to the default of 1. If  $0 < dAlphaMax < 1$ , the reported  $p$ -value is the minimum of the actual value and `dAlphaMax`. The smaller this is set, the less computation time is required.

- If `HFileIn` is omitted or set to 0, the usual bootstrap  $p$ -values are reported. Otherwise, size-corrected values are also reported, using the tabulations in the named file. This size-correction option is chiefly for use in simulation experiments. It is the user's responsibility to ensure this file is correctly constructed. The `Simulation` derived class can be used to generate the EDFs under  $H_0$ , written as `HfileOut`, but ensure that the arguments of `Settings` (except this one) match in the two runs.
- If `iMeth = -1`, the virtual function `Tables()` must return  $p$ -values for the statistics based on tabulations, and nothing else is computed. These 'nominal'  $p$ -values will be returned by default in the other cases, whenever the `Tables()` virtual function is present, and is not disabled by returning `-1`.

## Bootstrap::TestCall

`Bootstrap::TestCall(const mcData);`

`mcData`      in:  $T \times n$  matrix containing the data in columns.

*Return value*

If  $0 \leq \text{iMeth} \leq 3$  in `Settings`, an  $m \times p$  matrix, where  $p = 2, 3, 4$  or  $5$  as `iMeth = 0, 1, 2` or  $3$ . The first column contains the  $m \geq 1$  actual statistics returned by `Estimate`, and the remaining columns the associated bootstrap  $p$ -values specified by `iMeth`, in the order: regular, double, FD1, FD2.

If `iMeth = 4` in `Settings`, an  $m \times 14$  matrix. The first columns contains the  $m \geq 1$  actual statistics returned by `Estimate`, and the remaining columns these 13 quantiles of the associated bootstrap distributions, in ascending order: 0.01, 0.025, 0.05, 0.1, 0.2, 0.3, 0.5, 0.7, 0.8, 0.9, 0.95, 0.975 and 0.99.

*Description*

This function calls the test or tabulation procedures specified in the most recent call to `Settings`. The results are also printed to the console, unless printing is suppressed using the virtual function `PrintCall`.

## Bootstrap::Shocks

`Bootstrap::Shocks(const mcResids, const iExtra);`

`mcResids`      in:  $T \times n$  matrix containing residuals.

`iExtra`      in: int, number of extra shocks required.

*Return value*

A  $(T + \text{iExtra}) \times n$  matrix containing randomly resampled (with replacement) rows of `mcResids`.

*Description*

This is intended to be called from the user-supplied function `BootstrapSample`. `iExtra` can take either sign, and if `iExtra = 0`, the dimensions of the returned matrix are the same as `mcResids`. Note that for  $n > 1$ , complete rows are sampled so that the contemporaneous correlation structure of the series is preserved. To generate independent series, call `Shocks` repeatedly for each column in turn.

## Bootstrap::BlockShocks

```
Bootstrap::BlockShocks(const mcResids, const iExtra, const
cBlength);
```

mcResids      in:  $T \times n$  matrix containing residuals.

iExtra        in: int, number of extra shocks required.

cBlength      in: int, length of block for block bootstrap. Set to 1 for regular case, equivalent to Bootstrap::Shocks.

*Return value*

A  $(T + iExtra) \times n$  matrix containing randomly resampled (with replacement) blocks of length cBlength of rows of mcResids. The last block is of length less than cBlength unless  $(T + iExtra) / cBlength$  is an integer.

*Description* As for Bootstrap::Shocks

## Bootstrap::WildShocks

```
Bootstrap::WildShocks(const mcResids, const iType);
```

mcResids      in:  $T \times n$  matrix containing residuals.

iType         in: int = 0 for the Rademacher 2-point distribution (default);

$$\eta_t = \begin{cases} 1 & \text{with probability } 1/2 \\ -1 & \text{with probability } 1/2 \end{cases}$$

= 1 for the skewness corrected distribution;

$$\eta_t = \begin{cases} -(\sqrt{5}-1)/2 & \text{with probability } p = (\sqrt{5}+1)/(2\sqrt{5}) \\ (\sqrt{5}+1)/2 & \text{with probability } 1-p \end{cases}$$

*Return value*

A  $T \times n$  matrix containing randomly transformed rows of mcResids.

*Description*

This function is called similarly to Shocks, to implement the wild bootstrap. The rows of the returned matrix have the form  $u_t^* = \hat{u}_t \eta_t$  where  $(\hat{u}_1, \dots, \hat{u}_T)'$  is the matrix mcResids and  $(\eta_1, \dots, \eta_T)'$  is a vector of i.i.d. drawings from the 2-point distribution specified by iType. Note that in this case, the number of returned rows must be equal to the number of rows of mcResids. As with Shocks, the contemporaneous correlation structure of the series is preserved. To generate independent series, call Shocks repeatedly for each column in turn.

## Bootstrap::ReturnDist

```
Bootstrap::ReturnDist();
```

*Return value*

A  $m \times R$  matrix with columns containing the statistics computed by Estimate, each sorted in ascending order, where  $R$  is the number of bootstrap replications (cReps).

*Description*

Calling this function following a call to `TestCall` makes the full set of bootstrap replications available for further analysis. Note that when the double bootstrap or fast-double bootstrap is specified, these are the “first round” replications, as for the regular bootstrap; in other words, drawings from the model specified in the initial call to `LoadModel`.

*The following virtual functions must be supplied by the user.*

## **Bootstrap::Estimate**

```
virtual Bootstrap::Estimate(const mcData);
```

`mcData`        in:  $T \times n$  matrix containing a data set.

*Return value*

Should return a  $m \times 1$  vector,  $m \geq 1$ , containing bootstrap statistics. In the present version, only  $m = 1$  is allowed with the double bootstrap (`imeth = 1` or `3` in `Settings`).  $m > 1$  will generate an error message in these cases.

*Description*

This function is called once with the actual data to obtain the test statistics, and then in each bootstrap replication with data generated in `BootstrapSample`. While not a requirement, for best results a test statistic should be asymptotically pivotal, with a distribution not dependent on nuisance parameters in large samples.

## **Bootstrap::LoadModel**

```
virtual Bootstrap::LoadModel(const mcData);
```

`mcData`        in:  $T \times n$  matrix containing a data set.

*Return value*

A  $T \times n$  matrix of variables for resampling.

*Description*

This function is called once at the start of each run, except in the fast double bootstrap procedure, where it is called in each iteration for the bootstrap data set. To validate the test procedure the variables returned are assumed to be i.i.d., at least in large samples. In the purely nonparametric bootstrap with i.i.d. data, it could simply return `mcData` unchanged. More usually, it should estimate a model subject to the restrictions of the null hypothesis, and return the residuals. Any parameters estimated in the fitting procedure should be stored in static variables for later use by `BootstrapSample`.

## **Bootstrap::BootstrapSample**

```
virtual Bootstrap::BootstrapSample(const mcResids);
```

`mcResids`     in:  $T \times n$  matrix of variables returned by `LoadModel`, for resampling.

*Return value*

A  $T \times n$  matrix of bootstrap variables.

*Description*

To construct a bootstrap test, this function must generate a data set subject to the restrictions of the null hypothesis. This will usually be done by inverting the model estimated in `LoadModel`. The residuals can be generated by using one of the Ox random number generators, if the distribution is known, or by a call to `Shocks` or `WildShocks`, to obtain a set of residuals randomly resampled from `mcResids`. It can make use of parameters estimated in `LoadModel`, if these have been stored in static variables.

*The following virtual functions can be supplied optionally, if required.*

## **Bootstrap::StoreParams**

```
virtual Bootstrap::StoreParams();
```

*Return value*

Should return an array containing any estimated parameters and/or generated data, stored in static variables that will be overwritten by a call to `LoadModel`.

*Description*

The fast double bootstrap method makes random drawings from the parameter space by calling `LoadModel` with the bootstrap data at each replication. These parameters will overwrite the ones fitted from the actual data, which must therefore be stored for later retrieval.

## **Bootstrap::RetrieveParams**

```
virtual Bootstrap::RetrieveParams(const aStore);
```

`aStore`        in: the array created by `StoreParams`.

*No return value.*

*Description*

This function should rewrite the contents of `aStore`, which is the array you created in `StoreParams`, into their original locations in static variables.

## **Bootstrap::Tables**

```
virtual Bootstrap::Tables (const vActual)
```

`vActual`        in:  $m$ -vector of test statistics, as returned by `Estimate()`

*Return value*

*Either a matching vector of  $p$ -values, or  $-1$  to disable.*

*Description*

This function should usually pass its argument to one of the Ox probability functions to generate a nominal  $p$ -value. For example, it could contain the line

```
return 1 - probn(vActual);
```

Other relevant Ox functions include `probt`, `probchi` and `probf`. Remember that the rejection region is always assumed to be the upper tail of the statistic. Nominal  $p$  values are not reported if this function is either absent, or returns  $-1$ .

The main purpose of this function is to provide nominal or asymptotic  $p$ -values, for comparison purposes, in Monte Carlo experiments in conjunction with the `Simulation` class.

## Bootstrap::PrintCall

```
virtual Bootstrap::PrintCall(const bLine, ...);
```

`bLine` in: 1 to send a `cr/lf` so that next output is on a new line, 0 otherwise

`...` in: any type, items for printing (as in `print` or `println`).

*No return value.*

### Description

This function replaces the `Ox print` and `println` functions to allow output to be diverted to a text file, if this option is selected in `Settings`. It is used by the class, and can also be called from the user's program to direct output to the same file. By default, output is written to the console, but the function is virtual and can be replaced. To suppress output altogether, replace it by the empty function `{}`.

Note: formatted numerical output should be converted to a string using the `Ox sprintf` function, before sending to `PrintCall`.

### Example 1 (see Ex1.ox in the zip file)

The program tests the hypothesis that the mean of a random sample is zero, against two-sided and one-sided alternatives. The  $t$  statistic, absolute and signed, is used in combination with the regular bootstrap.

### Example 2 (see Ex2.ox in the zip file)

For the same data set as in Example 1, this program generates the quantiles of the bootstrap distributions of the sample mean, the standard deviation and the studentized mean.

### Notes:

1. The constructor function of your derived class should always call the constructor function of the base class, `Bootstrap`, to set the defaults.
2. Note the braces in the second argument of `Settings`, which is an array. These are required, even when  $m = 1$ .
3. The rejection region is always the upper tail of the null distribution. To implement tests which reject in the lower tail, change the sign of the statistic.
4. In Ex2, `LoadModel` returns the data set unchanged, but computes the sample mean. `s_vrParam` has to be initialised because `Estimate` is called first, with the actual data.

## The Simulation Class

`Simulation` is a class derived from `Bootstrap` that facilitates Monte Carlo experiments on bootstrap tests. It can be used to estimate the true size of tests, and compute powers and size-corrected powers against specified alternatives. It requires one additional virtual function, to generate the Monte Carlo draw and set up the replication.

## Simulation::Simulate

```
Simulation::Simulate(const vrCase);
```

```
Simulation::Simulate(const vrCase, const cMCReps, const  
cWrtFreq, const HFileOut);
```

**vrCase** in: row vector, containing settings to be passed to SetupReplication.

**cMCReps** in: int (optional) Number of Monte Carlo replications.

**cWrtFreq** in: int (optional) Frequency for writing the EDFs of bootstrap  $p$ -values to a file.

**HFileOut** in: string (optional) Name and path for file to contain the EDFs. The extension should indicate the type of file, see the Ox savemat function.

*No return value.*

### Description

This function sets up and runs an experiment.

- The tests to be performed are specified according to the most recent call to Bootstrap::Settings, which should normally be called before Simulate.
- If cMCReps = 0, it is reset to 1000 (default).
- If cWrtFreq = 0 (default) the EDFs are not written. In general, this option should be set > 0 when the null hypothesis is true. In this case the EDFs estimate the true size of the tests. Setting cWrtFreq < cMCReps allows saving intermediate results, so that data are not lost if the run is interrupted by power failure or user intervention.
- If cWrtFreq > 0 and HFileIn is given a value in Bootstrap::Settings, two output files are created. HFileOut contains the EDFs of the uncorrected  $p$ -values, and "c\_" ~ HFileOut contains the EDFs of the  $p$ -values corrected by HFileIn. If the run with  $H_0$  true is repeated, the corrected EDF should be uniformly distributed by construction, so these settings can be used to check the numerical accuracy of the bootstrap test.
- By default, HfileOut = "sim\_edf.mat". The EDFs are written in columns, by test statistic and then by test method. In other words,  $m$  blocks of 1, 2, 3 or 4 columns, depending on imeth in Settings. Write to Excel or Givewin files for convenience of plotting the EDFs. However, note that only the Ox savemat function is used, so the columns in .xls or .in7 files are not labelled informatively.

## Simulation::SetupReplication

```
virtual Simulation::SetupReplication(const vrCase);
```

**vrCase** in: row vector; the contents of vrCase, as passed to Simulate.

*Return value*

$T \times n$  matrix containing a generated data set.

### Description

This function must generate the data for the Monte Carlo draw. vrCase can be used, for example, in the context of a loop calling Simulate, allowing different cases of the alternative hypothesis to be simulated in a single run.



## Error Trapping

The Boolean member variable `m_bError` is declared in `Simulation`. Use this to trap errors, such as inversion or convergence failures in estimation, without aborting the whole run. If `m_bError` is set to `TRUE` in any derived function, a message is printed, the replication is discarded, and another drawing taken. The run as a whole will be aborted only if the number of aborted replications exceeds 10% of `cMCReps`.

### *Example 3* (See `Ex3.ox` in the zip file)

This simulates a significance test in a regression model with two regressors and 10 observations, using 1000 Monte Carlo replications of 199 bootstrap replications. The regular and fast-double bootstrap methods are invoked

Two simulations are run. The first experiment simulates the null hypothesis that regressor 2 has a coefficient of zero, and tabulates the EDFs of the  $p$ -values in the file `sim.xls`. Selected quantiles of the distribution (1%, 5%, 10%, 15%) are also sent to the text output.

The second experiment simulates a case of the alternative hypothesis, with the coefficient of regressor 2 equal to 0.5. The nominal and corrected powers are reported for the four significance levels, using the estimated EDFs from the first run to implement the size correction.

#### *Notes:*

1. The regressors are generated as independent  $N(5,2)$  variates in each Monte Carlo replication. However, they are treated as fixed in repeated samples so that, in particular, the same values are used in each bootstrap replication. These series are stored in a static variable by `SetupReplication`, for access from the supplied virtual functions.
2. The test statistic is the usual  $F$  statistic (equivalent to the squared  $t$ ), but the degrees-of-freedom terms are irrelevant in the bootstrap context, and are omitted to save computation—hence the designation "pseudo- $F$  stat."
3. The OLS estimates of the model under  $H_0$  are computed in `LoadModel` and stored in a static variable, where they can be accessed by `BootstrapSample`.
4. The parameters are saved using `StoreParams` and `RetrieveParams`, to allow implementation of the fast double bootstrap.
5. Including the double bootstrap in the set of tests (change `imeth` to 3) is slower, but perfectly feasible. Setting `cDReps` = 199 and `dAlphaMax` = 0.15, so that only  $p$ -values not exceeding this value are reported, the run time for this example increases only about 10-fold. If a more extreme alternative is chosen in Run 2, such that most  $p$ -values are near zero, the increase is only about 6-fold.

## References

- Beran, R. (1988) Prepivoting test statistics: a bootstrap view of asymptotic refinements, *Journal of the American Statistical Association* 83, 687-697.
- Davidson, J. (2003) Alternative Bootstrap Procedures for Testing Cointegration in Fractionally Integrated Processes. Working Paper at <http://www.cf.ac.uk/carbs/econ/davidsonje/altbootpr4.pdf>
- Davidson, R. and J. MacKinnon (2000) Improving the reliability of bootstrap tests. Working paper, at <http://les1.man.ac.uk/sapcourses/esgc/Papers2000/davidson.pdf>
- Horowitz, J. L. (1999). The Bootstrap. Chapter for *Handbook of Econometrics* Vol. 5 North-Holland Elsevier (forthcoming).
- Liu, R. Y. (1988). "Bootstrap procedures under some non-I.I.D. models," *Annals of Statistics* 16, 1696–1708.
- Mammen, E. (1993). "Bootstrap and wild bootstrap for high dimensional linear models," *Annals of Statistics* 21, 255–285.
- Nankervis, J. C. (2001) Stopping rules for double bootstrap tests. Working paper, University of Surrey.